

IDETC2020-19380

**ATTENTION ROUTING: TRACK-ASSIGNMENT DETAILED ROUTING USING
ATTENTION-BASED REINFORCEMENT LEARNING**

**HAIGUANG LIAO¹ QINGYI DONG¹ XULIANG DONG¹ WENTAI ZHANG¹
WANGYANG ZHANG² WEIYI QI² ELIAS FALLON² LEVENT BURAK KARA^{1*}**

1. Carnegie Mellon University
Pittsburgh, PA 15213
2. Cadence Design Systems
San Jose, CA 95134

ABSTRACT

In the physical design of integrated circuits, global and detailed routing are critical stages involving the determination of the interconnected paths of each net on a circuit while satisfying the design constraints. Existing actual routers as well as routability predictors either have to resort to expensive approaches that lead to high computational times, or use heuristics that do not generalize well. Even though new, learning-based routing methods have been proposed to address this need, requirements on labelled data and difficulties in addressing complex design rule constraints have limited their adoption in advanced technology node physical design problems. In this work, we propose a new router — attention router, which is the first attempt to solve the track-assignment detailed routing problem by applying reinforcement learning. Complex design rule constraints are encoded into the routing algorithm and an attention-model-based REINFORCE algorithm is applied to solve the most critical step in routing: sequencing device pairs to be routed. The attention router and its baseline genetic router are applied to solve different commercial advanced technologies analog circuits problem sets. The attention router demonstrates generalization ability to unseen problems and is also able to achieve more than 100× acceleration over the genetic router without severely compromising the routing solution quality. Increasing the number of training problems greatly improves the performance of attention router. We also discover a similarity between the at-

tention router and the baseline genetic router in terms of positive correlations in cost and routing patterns, which demonstrate the attention router's ability to be utilized not only as a detailed router but also as a predictor for routability and congestion.

1 INTRODUCTION

Integrated circuits (IC) are becoming increasingly more sophisticated keeping pace with Moore's Law [1]. To solve the increasingly more complex IC system design problems, new advanced electronic design automation (EDA) tools are needed to help engineers especially in the domain of advanced technology node (< 16 nm) IC designs. In the physical design flow of IC, a critical step is *routing*, where paths for connecting separate groups of devices are generated based on the locations of devices determined in the previous step of placement. To make the problem tractable, the routing problem is addressed in two stages: *global routing* and *detailed routing* [2]. While global routing aims to coarsely assign space resources used for routing, detailed routing generates the exact routes that connect electric components. The placement and routing, while applied sequentially, are interdependent: a good placement makes routing simpler, and quantitative routing measures can in turn be used to assess the quality of a placement solution.

To achieve successful and high quality IC physical designs, prior works have emphasized quality of routing [3–5] vs. speed of attaining a solution [6, 7], and have developed routability pre-

*Address all correspondence to this author.

diction algorithms [8, 9]. However, existing routing algorithms are primarily based on heuristic based methods which impose stringent constraints and therefore do not generalize well to unseen problems. Although, there have been several learning-based methods to improve the performance of routing algorithms [10–12], these approaches either only work as routability predictors or are hampered by limited generalization ability and the inability to account for complex design rule constraints, which are becoming increasingly sophisticated in advanced technology nodes IC design. As such, fast routing algorithms with strong generalization ability are urgently needed. In this work, we present *attention routing*, which is an application of attention-model based reinforcement learning (RL) model, to solve the track-assignment detailed routing problems on advanced node technologies problem sets. The routing algorithm is designed to encode the design rules into the track-assignment steps. The RL algorithm addresses one of the most critical steps in routing, which is determining the best order sequence of the set of device pairs to be routed, such that the overall solution quality is maximized. To the best of our knowledge, this work is the first attempt to solve the detailed routing problem using RL. The RL model is a policy gradient method based on attention model [13]. We describe our attention router and also a genetic router, which is based on genetic algorithms (GA). Both methods are tested on commercial advanced technology nodes IC problem sets, performance is compared and analyzed.

2 BACKGROUND and RELATED WORK

2.1 Width Spacing Pattern

In advanced technologies node, the manufacturing and design rule constraints (*e.g.* those due to multi-patterning) have significantly increased the complexity of the physical design task. As a result, it is becoming increasingly more challenging for layout designers to parse and memorize all the design rules. A further layer of abstraction is introduced to address this issue, namely the Width Spacing Pattern (WSP).

WSPs define a set of track patterns that consist of different width and spacing configurations for metal wires. By restricting the routes on the WSP rows and tracks, many design rules associated with full custom designs, including those concerning the spacing, minimum widths, and coloring rules can be avoided. In this work, as we attempt to solve the detailed routing problem for analog circuits in advanced technologies (FinFET), the routing strategies we present follow the design rule specifications through the adoption of WSP abstraction.

2.2 Track-Assignment Routing

As mentioned above, routing is typically divided into two stages: *global routing* and *detailed routing* [2]. In global routing, the routing resources are allocated into sub-regions (in the WSP

setting, rows) and the detailed router will then implement the planned routes, satisfying various constraints (*e.g.* open, short, design rule checkers (DRC)). However, when many routes are sequentially implemented, the two-step solution could result in undesirable detours for global routes that are planned to be straight [14]. Such issues can be addressed with a time-consuming rip-up and re-route strategy, which involves heuristics that depend on the routing style and technology requirements [15]. Another approach is to insert a track assignment step between the global and detailed routing stages that aims to solve the routing problem in a more hierarchical manner [16–20].

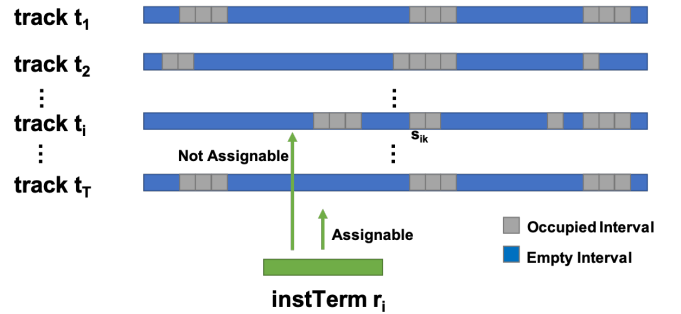


FIGURE 1: Schematic showing track assignment constraints.

The goal of track assignment is to place the long routes onto tracks defined by the WSP, with the constraints imposed by technology, routing resources, as well as conflicting routes being simultaneously considered. It also facilitates addressing layout dependent issues such as crosstalk [19]. After the long routes are embedded on the tracks, the detailed router’s job is to connect those components (instance terminals, *instTerms*) belonging to the same net, thereby significantly reducing the routing search space.

Let us denote t_i as the i -th track, T the set of tracks defined by the WSP, and s_{ik} the k -th occupied interval on track t_i , as shown in Fig. 1. Then, the utilized track resources on t_i is: $u_{t_i} = \bigcup_k s_{ik}$, and *instTerm* r_i is *assignable* to track t_j iff the respective track interval is not occupied ($r_i \cap u_{t_j} = \emptyset$). Similarly, the *instTerms* to be assigned are defined as, $r_i \in I$, where r_i is the i -th *instTerms* and I the *instTerm* set extracted from the global routing result. The task of assigning *instTerm* r_i onto track t_j , denoted as m_{ij} , is associated with an assignment cost C_{ij} , which reflects the cost including track occupation, perpendicular connection, via insertion [18]. The track-assignment step is therefore deciding a mapping M which assigns all the *instTerms* onto the available tracks without any conflict, while minimizing the assignment cost:

$$M^* = \min_M \left\{ \sum_{m_{ij} \in M} C_{ij} \right\} \quad (1)$$

s.t. $\forall m_{ij} \in M^*, r_i \cap u_j = \emptyset$

Note that this is a modified weighted bipartite matching problem, which is known to be NP-complete. As in [18], we solve it using a heuristic based algorithm, and the details are discussed in Section 3.1.

More specifically, in our case (as shown in Figure 2), the routing task consists of two sub-tasks, *i.e.* routing the instTerms on the appropriate tracks and connecting the instTerms. Although an instTerm could consist of many pins with different x-coordinates, an instTerm must be routed on a *single* track, making it suitable to use the track assignment formulation. Therefore, in the proposed approach, instTerm routing is addressed with track assignment and we then use attention-based RL model to solve the most critical part in actual routing the assigned inst-Terms.

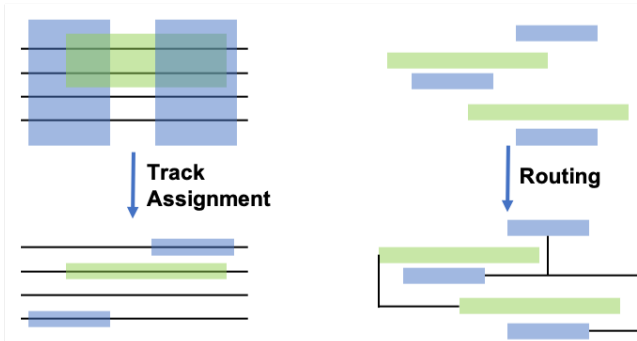


FIGURE 2: Schematic showing track assignment and routing.

2.3 Attention-based REINFORCE

Recent works [13] on using attention-model-based REINFORCE - a reinforcement learning (RL) algorithm to solve combinatorial problems have demonstrated near optimum performance with significant generalization capability compared to existing heuristic based method. It outperforms previous work including Pointer Network (PN) [21], actor-to-critic version of PN [22] and LSTM version of PN [23] in widely studied problem sets including Travelling Salesman Problem (TSP), Orienteering Problem (OP) and price collecting TSP. In solving these problems, the solution can always be formulated as a sequential decision process. One important reason they tend to be solved

reasonably well with reinforcement learning (RL) is that they can be modelled as Markov Decision Process (MDP).

In solving such combinatorial problems, the attention-model-based REINFORCE use an existing policy gradient RL model: REINFORCE. In a policy gradient RL algorithm, a model is used to learn a policy model $p(a|s, \theta)$, matching state s of a problem at each time step to a corresponding probability distribution of all actions a by iteratively optimizing the policy model parameters over training samples. The cost that training process aims to minimize is the expectation of reward r collected after certain policy p has been rolled out for an episode, which can be expressed as $E_p[r(\tau)]$ [24]. Following the policy gradient theorem, the gradient of the cost can be expressed as shown in Eqn. 2:

$$\nabla E_{\pi_\theta}[r(\tau)] = E_{\pi_\theta}[r(\tau) \left(\sum_{t=1}^T \nabla \log p_\theta(a_t | s_t) \right)] \quad (2)$$

which can be sampled and approximated from training data.

In REINFORCE, the above gradient of cost function is used to optimize or train the policy model. However, the training process of REINFORCE tends to be unstable due to the delayed reward mechanism of REINFORCE. Thus, REINFORCE with baseline is applied to stabilize REINFORCE.

In attention-model-based REINFORCE [13], the formulation of problem, taking Travelling Salesman Problem (TSP) as an example, can be described as follows: the solution is defined as a tour $\pi = (\pi_1, \dots, \pi_n)$, which is a sequence of the n nodes (cities) in a TSP problem s . The input of the policy model is based on the graph structure (layout of cities) of the TSP, and the policy model output a probability distribution $p_\theta(\pi_t | s, \pi_{1:t-1})$ over all the nodes that are likely to be visited at the next time step n , during which nodes already visited are masked to ensure zero probability to be visited. Based on this, a problem policy $p(\pi | s)$ is defined as Eqn. 3, which is the product of probability distribution for the n steps. If at each time step, the node with the highest probability is chosen, a solution (path) is said to be given in a deterministic greedy rollout manner:

$$p_\theta(\pi | s) = \prod_{t=1}^n p_\theta(\pi_t | s, \pi_{1:t-1}) \quad (3)$$

Based on the problem policy and policy gradient theorem, the gradient of the loss function used in the attention model is defined as Eqn. 4. For a TSP problem, the loss term $L(\pi)$ is the total tour length of the TSP problem instance. When applying the attention-based model to solve detailed routing problems, the loss is changed accordingly:

$$\nabla L(\boldsymbol{\theta}|s) = E_{p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)}[(L(\boldsymbol{\pi}) - b(s))\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)] \quad (4)$$

A rollout baseline $b(s)$ is applied that is periodically updated in the following ways: $b(s)$ is the cost of a solution from a deterministic greedy rollout of the policy defined by the best model so far. In actual implementation, a t -test is applied to ensure that the baseline is based on the best model so far.

The policy model is realized with attention-based encoder-decoder model which can be considered as a Graph Attention Network [25], as shown in Fig. 3. The encoder is similar to the Transformer architecture [26] encoder. After a first learned linear projection layer, the major part of encoder consists of N attention layers, with each layer consisting of two sublayers: a multi-head attention (MHA) layer and a fully connected feed-forward (FF) layer, skip-connection [27] and batch normalization (BN) [28] are applied at each sublayer. Formally, the attention layers can be expressed as follows:

$$\hat{h}_i = BN^l(h_i^{(l-1)} + MHA_i^l(h_1^{(l-1)}, \dots, h_n^{(l-1)})) \quad (5)$$

$$h_i^{(l)} = BN^l(\hat{h}_i + FF^l(\hat{h}_i)) \quad (6)$$

where $\hat{h}_i^{(l)}$ represents the output values (in vector form) in the i th node of l th MHA layer and h_i represents the output values of the $\hat{h}_i^{(l)}$ after BN.

At the heart of the MHA structure is the attention mechanism which can be summarized as a weighted message passing between the nodes in a graph. The weight of a message *value* that a node receives from a neighbor depends on the *compatibility* of its *query* with the *key* of the neighbor [13]. For each node, its corresponding *key*, *query* and *value* is obtained by projecting the node embedding/input by parameter matrices correspondingly, whose weights are automatically learned during training. The decoder consists of only one layer of MHA and it outputs the probability distribution of nodes to be visited at each time step based on the embeddings from the encoder and the output generated at the previous time steps.

Inspired by the similarity between existing combinatorial problems and the critical sequencing step underlying detailed routing algorithms, we propose an attention-based REINFORCE model as a new way to address the detailed routing problem. In applying the attention-based model to solve detailed routing, while the model architecture remains the same, the loss function

is modified and the definition of a node now becomes a pair of *instTerms*, which will be addressed in details Method. (Another motivation for applying learning-based algorithm to solve detailed routing is the difficulty to apply a robust heuristics based or hard-coded method to the sequencing step. It is worth mentioning that although there exists simpler algorithms [13] for standard combinatorial problems such as Nearest Neighbors for TSP, the unique hierarchical nature and complexity of IC physical design flow and routing makes these algorithms not readily applicable to solve detailed routing [12].)

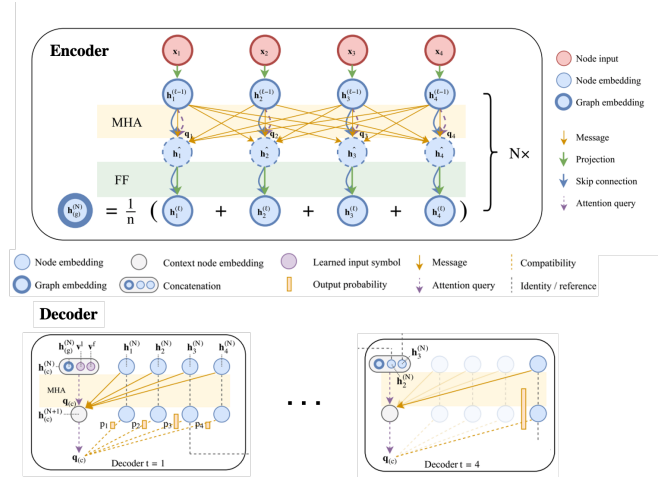


FIGURE 3: Encoder-decoder model structures. Adopted from [13].

2.4 Genetic Algorithm

Genetic algorithms (GA) [29] have been widely used to solve combinatorial optimization problems [30, 31]. It is realized with iterations of generations, as schematically illustrated in Fig. 4. For one generation, a *population* consisting a pool of chromosome is firstly generated either randomly or from elite parents of previous generation. A *fitness* function is then applied to calculate the fitness of each chromosome in the population, a proportion of the chromosome in the original population that has higher fitness scores are selected to be the *elites*, which naturally becomes parents for generating the next generation of population. A new generation of population is generated by *crossover* and *mutation* operations of chromosome among elites. In the next generation iteration, the previous population is replaced by newly generated population.

In this work, GA algorithm, as a comparison to attention model, is used in genetic router for determining the sequence of *instTerm* pairs to be routed, which significantly determines the quality of detailed routing solutions. GA has been one of the

best methods in solving combinatorial optimization problems in IC physical design [32–34], especially when no other heuristics and learning methods are not readily available. Unfortunately, although it tends to work well in small scale problems with no stringent run time requirements, it suffers from lack of generalization ability, large run time cost and limited scalability. In this work, we propose to use attention router as an alternative to the genetic router in track-assignment detailed routing.

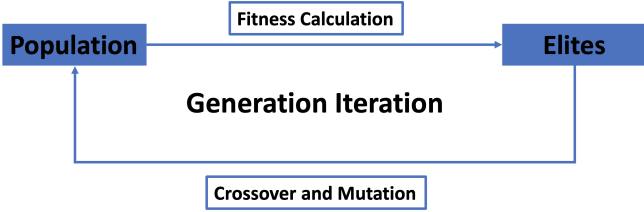


FIGURE 4: Generation iteration of genetic algorithm (GA).

3 METHOD

Fig. 5 illustrates our attention router model. Firstly, all problem files in a specific problem set describing the design information including instTerms locations and nets information is read in and parsed by the *Initializer*. *Track Assigner* is then applied to complete the track assignments for all instTerms in each problem. Once complete, the exact locations of all instTerms are determined. Next, *Pin Decomposer* is then applied to all the nets of each problem to further simplify each problem for the subsequent routing in the form of a set of instTerm pairs.

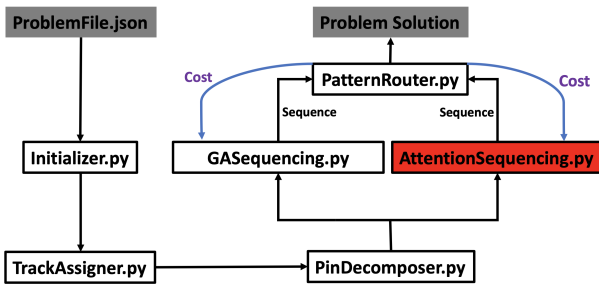


FIGURE 5: Pipeline of our attention routing.

In routing instTerm pairs, GA (for genetic router) and

attention-based REINFORCE model (for attention router) are applied to determine the sequence of the instTerm pairs in a problem to be routed. *GA Sequencing* is executed only once, while *Attention Sequencing* is firstly trained by solving problems from the training set, and then applied to problems in both training and test sets. During execution of *GA Sequencing* and *Attention Sequencing*, the same *Pattern Router* function is utilized to compute the exact routes connecting each instTerm pairs and to calculate the cost of the solution. The problem solution is a concatenation of the actual routes for all the instTerm pairs and the cost of a solution is defined as a weighted sum of *wirelength* (WL) and *number of openings* (#Open), which is the number of instTerms pairs that remain unconnected due to a lack of feasible route for the given problem. The cost is given in Eqn. 7. Since openings is highly undesirable in the physical design process, in this research weights are set as: $w_1 = 1, w_2 = 10$

$$Cost = w_1 * WL + w_2 * \#Open \quad (7)$$

The details of the individual modules of our method are provided next. Python and Pytorch (Machine Learning Framework) are used for implementing the proposed algorithm.

3.1 Track Assigner

The first step of routing is to assign instTerms to WSP tracks. As the x-coordinates of all instTerms are fixed, the routing problem is reduced to finding the appropriate track while satisfying the assignment rules and ensuring no short circuits.

Not all tracks can be used to route an instTerm. For illustration, in Fig 6 (a) with three tracks, Track 1 and 2 can only contain gate (G) and source/drain (S/D) terminals, while Track 3 can have both G and S/D terminals. Specifically, in this work, we have seven tracks per row, where the G terminals should be on tracks 1, 2, 6, and 7, S/D terminals on tracks 2, 3, 4, 5, and 6, hence for instTerms composed of both G and S/D, only track 2 and 6 shall be used.

Two graphs are used in the track assigner, namely the *overlap graph* and the *assignment graph* (Fig 6). The overlap graph models the conflicts between instTerms, where each node represents an instTerm, and an edge exists between two nodes if they belong to different nets and their x-ranges overlap (implying that they cannot be assigned to the same track). This constitutes a horizontal constraint graph [2]. The assignment graph is a weighted bipartite graph, the nodes on one side are the instTerms and the other are the available tracks, if an instTerm is assignable to a track, these two nodes are connected by an edge with a weight as the assignment cost. Because instTerm is constrained to route on the tracks of the containing row, the vertical connection and via costs can be omitted, and we model the cost considering only the horizontal track utilization.

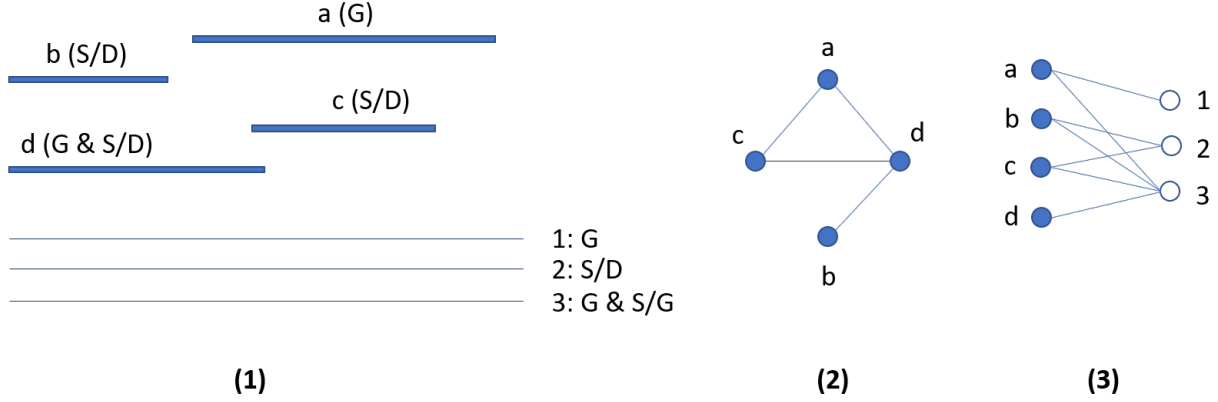


FIGURE 6: Illustration of the track assignment problem: (1) the instTerms and tracks, (2) the overlap graph, (3) the assignment graph.

With the help of the two graphs, the track assignment problem is reduced to matching the instTerm nodes to the track nodes in the assignment graph while minimizing the matching cost, such that no conflicting instTerm nodes in the overlap graph are matched to the same track. For the standard bipartite matching problem, [35] provides a polynomial algorithm, but the problem now becomes NP-complete [18] after introducing the assignment conflict constraint. As instTerm splitting is not allowed, we modified the algorithm presented in [18] to solve the instTerm track assignment problem, and the algorithm is described in Algorithm 2.

Algorithm 1: The track assignment algorithm.

Input : Netlist containing the instTerm and track information

Output: Assigned instTerm-track pairs

- 1 Build overlap graph $G_O = (V_o, E_o)$ and assignment graph $G_A = (V_A, E_A, w)$;
 - 2 **while** *Exists assignable instTerms* **do**
 - 3 Find the largest clique K_m in G_O ;
 - 4 Perform weighted bipartite matching on the sub-graph $G_m = (V_m, E_m)$, where $V_m \subseteq V_A, E_m \subseteq E_A, V_m \in K_m$;
 - 5 Assign the uniquely assignable instTerms to the corresponding track (look-ahead heuristic in [18]);
 - 6 Update G_O and G_A : remove assigned instTerm nodes and associated edges;
 - 7 **end**
-

3.2 Pin Decomposer

Each net is composed of multiple instTerms. Each instTerm has the coordinate of (x_1, x_2, y) , $x_1, x_2, y \in \mathbf{Z}^+$. In order to simplify the problem, instead of directly working on the sequence of nets, we first decompose each net into multiple two-instTerm pairs, so that after the decomposition, our model will produce the best sequence of these instTerm pairs. Kruskal’s algorithm [36] is utilized to construct a Minimum Spanning Tree (MST) first, as the MST naturally reveals the pin pairs that should be connected as shown in Fig. 7a.

In order to create the MST, a distance matrix is needed, where each element (i, j) in this matrix the distance between instTerm i and instTerm j . However, due to the fact that we are dealing with instTerms (bars) instead of nodes (points), distance is computed as the minimum Manhattan distance between the instTerm bars as shown in Fig. 7b and Fig. 7c. Note that even after this decomposition, we are still dealing with instTerm pairs rather than node pairs.

3.3 Pattern Router

We route each instTerm pair sequentially using a simplified pattern router [37] in the rectilinear space that can be modelled as a graph $G(V, E)$. In the graph, each edge has a capacity c_{ij} , which is initialized to 1 before routing. In routing the instTerm pairs of a problem, the routable paths are edges e_{ij} , $i, j \in \mathbf{Z}^+$ with non-zero capacity.

3.3.1 Routing two vertices We loop through all combinations of (v_i, v_j) , where v_i is a vertex from instTerm i , and v_j is a vertex from instTerm j . For each combination, we use our simplified pattern router, where we only consider “L” patterns and then “Z” patterns if “L” patterns fail.

“L” pattern routing There are 2 kinds of “L” patterns: upper “L” and lower “L”, which are shown in Fig. 8a and Fig. 8b

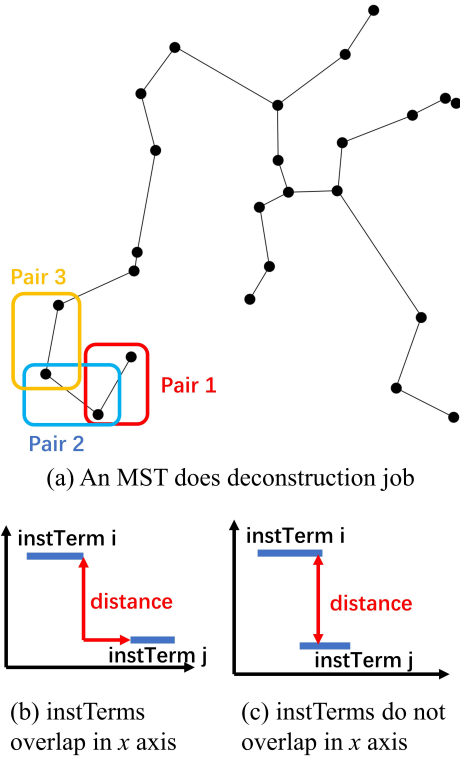


FIGURE 7: Pin decomposer. (a) An MST reveals in the instTerm pairs. (b)(c) Calculation of the distance between two instTerms.

respectively. Straight lines are also considered as a special case of “L” pattern, when two instTerms overlap in the x axis or share the same y coordinate.

“Z” pattern routing If “L” patterns fail, our router employs “Z” pattern routing. There are also two kinds of “Z” patterns, as shown in Fig. 8c and Fig. 8d.

If no above patterns can route (v_i, v_j) , an opening occurs.

3.3.2 Post processing Once v_i and v_j are routed successfully, we obtain the path that connects them. Any redundancies in the path are removed. Fig. 8e and Fig. 8f illustrate that the redundancy is the overlapping parts between the path and instTerms i, j .

3.4 Attention Model Implementation

In order to find an optimized routing solution that minimizes the loss among all possible routing sequences, we use an attention based encoder-decoder model with a rollout baseline. We define each problem instance as a graph with n nodes, and each node $n_i, i \in 1, \dots, n$ is represented by an instTerm pair between instTerm i and instTerm j . Each instTerm pair is in the form:

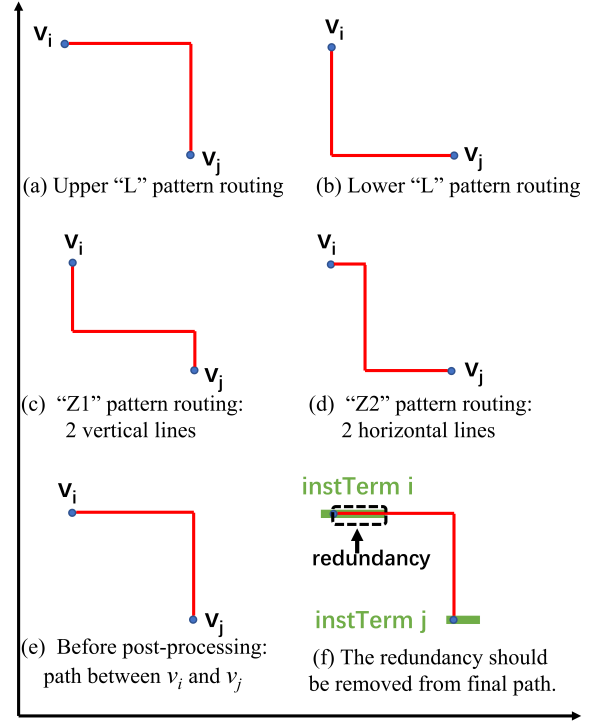


FIGURE 8: Two pin router. (a)(b) “L” pattern routing, (c)(d) “Z” pattern routing, (e)(f) Post Processing.

$$(x_{i1}, x_{i2}, y_i, x_{j1}, x_{j2}, y_j, l), \forall i, j, i \neq j,$$

where x_{i1}, x_{i2}, y_i represents the xy -coordinates of instTerm i . Similarly, x_{j1}, x_{j2}, y_j represents the xy -coordinate of instTerm j ; l represents the net index which is later used in the routing part and thus will not be discussed in detail here. We define the solution to the routing sequence π as a permutation of the n nodes.

Since the number of instTerms varies in each routing problem, to ensure that each problem instance s has the same number of nodes n (inst pairs), we perform two possible padding strategies on the problem instance: Pad Random and Pad Empty. In the Pad Random strategy, we uniformly sample xy -coordinates in the domain of all instTerms; in the Pad Empty strategy, we are not concerned with the actual coordinates and instead pad with all-zero nodes in the form of $(0, 0, 0, 0, 0, 0)$. In each strategy, we set the graph size n to be the maximum graph size of all the problem instances, and pad nodes to problem instances of smaller size. After experiments with the two padding strategies, we decide to use the Pad Empty strategy, which performs more stably and generates routing sequences with smaller loss. We think the result of the Pad Random strategy can be improved if the coordinate sampling is done based on the original coordinate distribution of the instTerms instead of a uniform sampling.

After careful parameter tuning, we set the number of training batches $B = 20$, and for each batch, the training batch size $T = 5$.

We train our model on epoch sizes $E = 100$.

Our current problem sets contains different total number of problem instances, which we split into 60% as training, 20% as validation, and 20% as test cases. Each epoch is a walkthrough of all the problem instances in the training set. In each batch, the dataset loader loads 5 out of the training problem instances in sequence as the current training batch. At the end of each epoch, the model is evaluated on the validation set, and the average loss is computed. After completion of all epochs, the model with the smallest average loss is used to evaluate the test set by generating corresponding routing sequences to each test problem.

We define the loss $L(\theta|s) = E_{p_\theta(\pi|s)}[L(\pi)]$, where $L(\pi)$ is a vector of length 5, containing losses of each problem instance returned by the pattern router as discussed in section 3.4. The loss of each problem instance returned by the pattern router is defined as a weighted sum of the total wire length and number of openings using the routing sequence π generated by the attention model, as shown in Eqn. 7. We optimize the loss $L(\theta|s)$ by gradient descent, using the REINFORCE gradient estimator with rollout baseline $b(s)$ [13]:

$$\nabla L(\theta|s) = \mathbf{E}_{p_\theta(\pi|s)} [(L(\pi) - b(s)) \nabla \log p_\theta(\pi|s)] \quad (8)$$

At the end of each epoch, we perform a one-sided t-test between the current model and the baseline model with a significance parameter α to decide whether or not the baseline model should be updated. The algorithm is described in Algorithm 2.

Algorithm 2: Attention Sequencing

Input : Number of epochs E , batch size B , training set T , significance α

Output: Sequence based on best policy

```

1 Init  $\theta, \theta^{BL} \leftarrow \theta$ ;
2 for  $epoch=1, \dots, E$  do
3   for  $batch=1, \dots, B$  do
4      $t_i \leftarrow \text{SampleInstance}() \forall i \in 1, \dots, T$ ;
5      $\pi_i \leftarrow \text{SampleRollout}(t_i, p_\theta) \forall i \in 1, \dots, T$ ;
6      $\pi_i^{BL} \leftarrow \text{GreedyRollout}(t_i, p_{\theta^{BL}}) \forall i \in 1, \dots, T$ ;
7      $\Delta L \leftarrow \sum_{i=1}^B (L(\pi_i) - L(\pi_i^{BL})) \Delta \theta \log p_\theta(\pi_i)$ ;
8      $\theta \leftarrow \text{Adam}(\theta, \Delta L)$ ;
9   end
10  if  $\text{OneSidedPairedTTest}(p_\theta, p_{\theta^{BL}}) \leq \alpha$  then
11     $\theta^{BL} \leftarrow \theta$ ;
12  end
13 end
```

3.5 Genetic Algorithm (GA) Sequencing

The GA-based sequencing, which works as a comparison to attention-based model in this work follows the typical generation iterations of the GA algorithm shown in Fig. 4, with crossover and mutation operations within each generation. The details of the GA sequencing are shown in Algorithm 3. In this problem, each chromosome consists of an ordered vector of numbers, representing the routing sequence for all instTerms pairs in a problem. Since we are trying to minimize the cost in Eqn. 7, the fitness of a chromosome is the negative value of the cost in Eqn. 7 by solving the corresponding problem with the sequence indicated by the chromosome. Model parameters for the GA sequencing are set as: *generation number*: 10, *population size*: 10, *elites size*: 4, *number of mutations*: 1. Note that a limited number of generations is chosen to avoid the run time of GA sequencing from becoming too long.

Since each chromosome in our algorithm is a sequence rather than independent numbers, each number can only appear once a chromosome. To address this uniqueness, the crossover and mutation operations adopted in research is demonstrated in Fig. 9. In generating a new child’s chromosome, partially matched crossover is adopted. After crossover, a newly generated chromosome is obtained. In the mutation step, two genes in two random selected locations in the newly generated chromosome switch their positions. This crossover and mutation method guarantee that all generated kids represent a legal sequence.

Algorithm 3: Genetic Algorithm Sequencing

Input : Number of generations G , population size P , elites size Q , number of mutations M

Output: Last generation of sequencing (chromosome)

```

1 Init chromosomes  $\{C_1, \dots, C_P\}$  in first generation;
2 for  $generation=1, \dots, G$  do
3   Select elites  $E_1, \dots, E_Q$  based on Fitness Score;
4   for  $i=1, \dots, P$  do
5      $C_i \leftarrow \text{CrossOver}(E_i, E_j) i, j \in 1, \dots, Q$ ;
6      $C_i \leftarrow \text{Mutation}(C_i)$ ;
7   end
8 end
```

4 EXPERIMENTS

In order to assess the performance of the attention router and its comparison to the baseline genetic router, both algorithms are applied to detailed routing problems from two problem sets: *Small* and *Large*, which are both analog design problems based on commercial advanced node technologies (sub-16 nm technology). To be specific, *Small* problem set consists of different

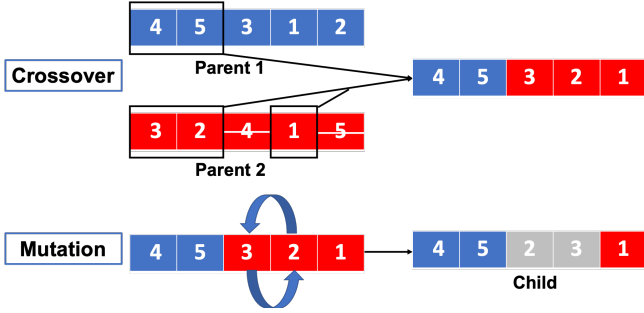


FIGURE 9: Crossover and mutation methods used in the GA sequencing.

placement solutions for Comparators and OpAmp, while *Large* problem set consists of different placement solutions of Analog-to-Digital Converter (ADC). For *Small* problem set, the number of instTerms for each problem range from 10 to 100, and in *Large* problem sets, the number of instTerms for each problem range from 100 to 1000.

We ran experiments based on the two problem sets. Three experiments were conducted using the *Small* data set with 100, 500, and 5000 training problems denoted as *Small100*, *Small500*, and *Small5000* respectively. Two experiments were conducted using the *Large* data set with 100 and 500 training problems denoted as *Large100* and *Large500* respectively. In the genetic router, GA Sequencing is run for each of the problems in the problem sets. In the attention router, attention sequencing is trained iteratively using the training sets and then applied to previously unseen problems in the test sets. For the four sets of experiments, the key parameters for the attention model are: *batch size*= 5 and *epoch number*= 100. Increasing the batch size and the number of epochs significantly improves the attention model’s performance, (thus we set the epoch number to allow the model gain enough learning experience, while not spending too much time for training.). All experiments are run on a workstation with an Intel Core i7-6850 CPU . In training the attention router, it takes around 6 minutes for a training epoch on problem sets *Small500* and around 25 minutes for a training epoch on problem sets *Large500* .

5 RESULTS AND DISCUSSIONS

5.1 Training

Fig. 10 shows cost versus training epochs plots for problem sets *Small100* and *Small500*. The high variation during the training process is an intrinsic property of the REINFORCE policy gradient algorithm used in the attention model. It can be explained by the “delayed reward” of policy gradient REINFORCE algorithm used to optimize the network, as shown in Eqn. 2. In the equation, the reward signal is not obtained until T steps of actions $\{a_1, \dots, a_T\}$ have been taken, then it is multiplied with

the summation of log-values of $p_\theta(a_t|s_t)$ at each step to form the gradient values for optimizing the policy networks. The delayed reward signal mechanism makes the training unstable as there is no clear guidance in terms of each action’s contribution in the action sequence $\{a_1, \dots, a_T\}$ to the reward. As such, the gradient based on the policy gradient theorem equation in Eqn. 2 can only optimize the policy networks with a rough guidance in terms of the optimization directions, instead of a more desirable one that can lead to monotonically decreasing cost values.

The variation in the training process of REINFORCE is remedied with the introduction of a baseline [24], which can be described as a gauge for the difficulties of problem the model is solving and usually leads to faster learning in the REINFORCE model. In this work, although a baseline has been implemented, the variation in training is still present, which might be further reduced with techniques such as decay learning rate and application of critic networks [13], which will be a part of our future work.

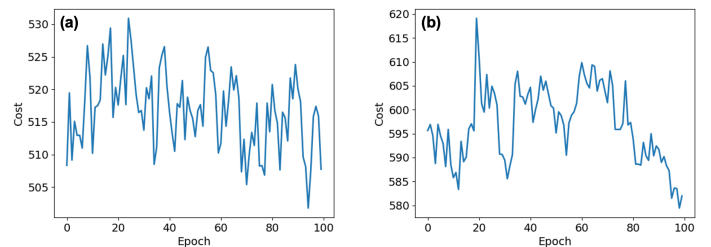


FIGURE 10: Cost vs. training epochs on (a) *Small100*, and (b) *Small500* sets.

5.2 Attention router performance

5.2.1 Performance comparison between attention model and GA. Figure 11 shows the cost comparison between the attention router and the genetic router for all problems in the training and test sets for *Small500* and *Large500* problems. In each figure, the horizontal axis corresponds to problem indices sorted based on an ascending order of the genetic router results. For problem sets *Small500*, while the genetic router performs better in almost all problems compared to the attention router, the difference in cost for a given problem between the two routers is rather small (mostly within 100). For problem sets *Large500*, which has approximately ten times the number of instTerms than *Small500* in each problem, while the genetic router still performs better overall, the number of cases in which the attention router outperforms GA is higher. It has been argued in prior work [38] that when applying a genetic router to solve large scale problems, it tends to exhibit high computational cost

accompanied by a degradation of the solution quality. This is primarily due to the increased complexity of problems, where the number of possible sequences of n instTerm pairs is $O(n!)$. This makes increasingly larger problems intractable for GA with limited computational cost.

By comparing the performance of the attention router on training and test sets (as shown in Fig. 11), it can be seen that in both problem sets, the performance of the attention router is similar in training sets and test sets. This implies that the attention router can solve previously unseen problems once it is trained on the training set. This is due to the attention model’s ability to learn proper strategies across various spatial structures of the instTerm pairs. The multi-head attention (MHA) mechanism can be seen as a message passing method that allows each instTerm pair to communicate with all other instTerm pairs in the same problem regarding their relative spatial information [13]. In this way, an instTerm pair’s spatial configuration within a space shared by other instTerm pairs can be continuously monitored and factored in. This spatial information is then used to form the sequential decisions for the final sequencing of the instTerm pairs.

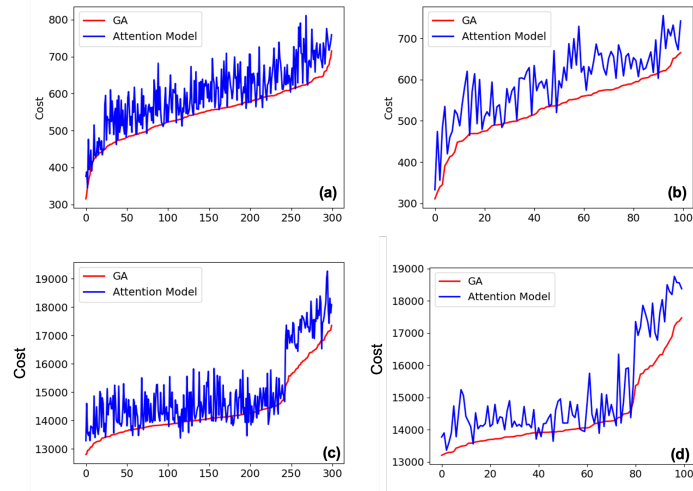


FIGURE 11: Cost comparison across different problems: (a) *Small500* training, (b) *Small500* test, (c) *Large500* training, (d) *Large500* test.

Figure 12 compares the cost vs. run time of the attention router and the genetic router on problem sets *Small500* and *Large500*. Attention router’s results are shown in blue dots, while genetic router’s results are shown in red dots. In both plots, while the cost range of attention router’s results is slightly higher than the genetic router ones, the run time of attention router is more than two orders of magnitude ($100\times$) shorter than the

genetic router: For *Small500* problem set, genetic router takes more than 10 seconds to solve each problem, while for the attention router, the time is less than 0.1 seconds. For the *Large500* problem set, the genetic router takes close to 100 seconds for each problem, while the attention router only takes a little more than 0.1 seconds to solve a problem.

This significant increase in speed enabled by the attention router is due to different algorithmic structures of the attention model and GA. For the attention model, once the model’s training is completed on the training set in an off-line setting, it is applied to new problems in a forward fashion through primarily matrix multiplications without iterations. The genetic router, on the other hand, solves each problem anew, without the ability to learn from previously solved problems. The significant run-time acceleration enabled by the attention router provides a new alternative for the GA router especially in the early stages of the IC design where placement decisions are yet to be made in the upstream of the workflow. In such instances, the inner optimization involving detailed routing can be significantly accelerated using the attention model as a way to provide useful guidance to the placement algorithm, by leveraging the positive correlations between results from the attention model and GA (Fig. 13). However, as our results suggest, for the ultimate detailed routing decisions, the genetic router currently provides better quality solutions (Fig. 11).

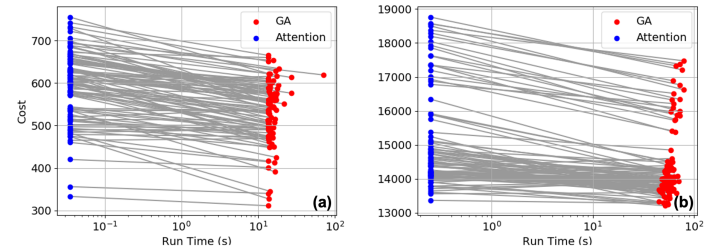


FIGURE 12: Cost vs. run time on test sets for problems in (a) *Small500*, (b) *Large500*. The same problems are connected with gray lines.

The routability prediction is crucial in the placement step of IC physical design [39,40]. In order to achieve successful design of a chip, there always exists the need in placement step to fast and accurately assess whether there is a good routing solution exists based on certain placement solution. Existing routability prediction algorithms [39–42] have been mainly focusing on global routing stage, and even those that takes into account detailed routing stage [39], supervised learning method is used, which makes it depend on other routers to provide labelled data. The attention router in this research provides a promising way

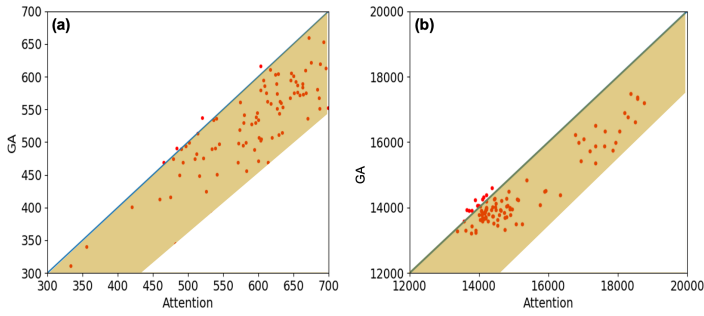


FIGURE 13: Cost comparison between attention model and GA in (a) *Small500*, (b) *Large500*.

for routability prediction by leveraging its positive correlations with genetic router solutions, which is a feasible solution that can be utilized for providing high quality solutions for detailed routing. Another advantage is that, since the attention router utilizes RL, it does not rely on any supervised learning requiring labelled training data. Yet, in order to assess the accuracy of routability prediction with the use of attention router, the model needs to be tested on more problems across different problem sets, which will be part of our future work.

5.2.2 Effect of Training Sample Numbers on the Performance of Attention Model. To investigate the effect of training sample numbers on the performance of the attention model, the model is trained on the problem set *Small5000*. Fig. 14 shows the results of the trained model performance on the training and test sets. Compared to the model trained on *Small500* (Fig. 11a and b), the solutions produced by the attention router improves significantly, approaching the genetic router’s solutions. The sample efficiency of the attention router remains part of our future work.

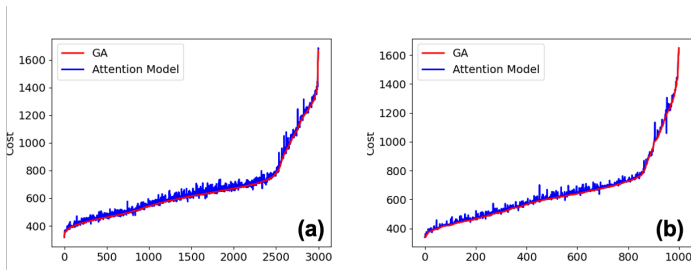


FIGURE 14: Cost comparison with increased number of training problems : (a) *Small5000* training, (b) *Small5000* test.

5.2.3 Final Routes The routing results of the attention router and the genetic router on a randomly chosen problem from the *Large500* problem set are shown in Fig. 15. Black dots and bars correspond to instTerms and the colored lines represent the actual routes. As seen, the solutions of the two routers share some similarity. For instance, high congestion regions (shown in red circles indicating densely configured routes) are located at similar regions of the physical space. The blow out regions of the attention router (c,d) and the genetic router (e,f) of the same area also indicate similar patterns in routes and similar unconnected instTerms. This type of similarity suggests that the attention router can be used by upstream modules to rapidly predict congestion regions as well as the instTerms that may remain open in the detailed routing stages.

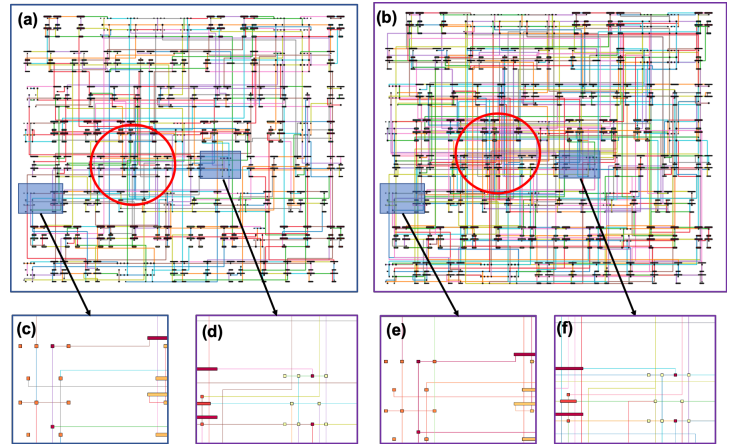


FIGURE 15: Final routes: (a) Attention model and (b) GA on a problem in *Large500* problem set; magnified routes on same regions of (c,d) attention model solution and (e,f) GA solution.

6 CONCLUSIONS

We present a new approach to the track-assignment detailed routing using RL. A detailed routing pipeline we call the attention router that takes into account complex design rule constraints is developed and the attention-model based REINFORCE algorithm is applied for the ordering of instTerm pairs. The attention router and a baseline genetic router is tested on different commercial advanced technologies analog circuits problem sets. The attention router demonstrates a generalization ability to unseen problems from the same problem set after appropriate training. While the genetic router can have slightly better quality solutions, the attention router is able to achieve more than $100\times$ acceleration compared to the genetic router without a severe degradation of the routing solution. Increasing the number

of training problems also greatly improves the performance of the attention router on both training and test sets. Positive correlations in terms of cost are also found between the attention router and the genetic router, which enable the possibility of applying attention router as a routability predictor in the placement stage. Similarities in the routing solution patterns (congestion region and disconnected instTerms locations) are also discovered, which demonstrate the attention router’s ability to work as a more fine-grained congestion predictor and a predictor for disconnected instTerms locations in detailed routing. Future work includes analyzing the correlation between the attention router performance and the genetic router in terms of cost and congestion. Sample efficiency of the model will also be studied to provide guidance on training set size when solving problems of different sizes.

7 ACKNOWLEDGEMENTS

This work is funded by the DARPA IDEA program (HR0011-18-3-0010; Funder ID: 10.13039/100006502). The authors would like to thank Prof. Barnabas Póczos for his useful feedback.

REFERENCES

- [1] Schaller, R. R., 1997. “Moore’s law: past, present and future”. *IEEE spectrum*, **34**(6), pp. 52–59.
- [2] Sherwani, N. A., 2012. *Algorithms for VLSI physical design automation*. Springer Science & Business Media.
- [3] Hu, J., and Sapatnekar, S. S., 2001. “A survey on multi-net global routing for integrated circuits”. *Integration*, **31**(1), pp. 1–49.
- [4] Mo, F., Tabbara, A., and Brayton, R. K., 2001. “A force-directed maze router”. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*, IEEE, pp. 404–407.
- [5] Cong, J., Kahng, A. B., Robins, G., Sarrafzadeh, M., and Wong, C.-K., 1992. “Provably good performance-driven global routing”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **11**(6), pp. 739–752.
- [6] Soukup, J., 1978. “Fast maze router”. In *Proceedings of the 15th Design Automation Conference*, IEEE Press, pp. 100–102.
- [7] Chang, Y.-J., Lee, Y.-T., and Wang, T.-C., 2008. “Nth-route 2.0: a fast and stable global router”. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, IEEE, pp. 338–343.
- [8] Li, W., and Banerji, D. K., 1999. “Routability prediction for hierarchical fpgas”. In *Proceedings Ninth Great Lakes Symposium on VLSI*, IEEE, pp. 256–259.
- [9] Pui, C.-W., Chen, G., Ma, Y., Young, E. F., and Yu, B., 2017. “Clock-aware ultrascale fpga placement with machine learning routability prediction”. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, pp. 929–936.
- [10] Qi, Z., Cai, Y., and Zhou, Q., 2014. “Accurate prediction of detailed routing congestion using supervised data learning”. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, IEEE, pp. 97–103.
- [11] Tabrizi, A. F., Rakai, L., Darav, N. K., Bustany, I., Behjat, L., Xu, S., and Kennings, A., 2018. “A machine learning framework to identify detailed routing short violations from a placed netlist”. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6.
- [12] Liao, H., Zhang, W., Dong, X., Póczos, B., Shimada, K., and Burak Kara, L., 2020. “A deep reinforcement learning approach for global routing”. *Journal of Mechanical Design*, **142**(6).
- [13] Kool, W., Van Hoof, H., and Welling, M., 2018. “Attention, learn to solve routing problems!”. *arXiv preprint arXiv:1803.08475*.
- [14] Hetzel, A., 1998. “A sequential detailed router for huge grid graphs”. In *Proceedings Design, Automation and Test in Europe*, IEEE, pp. 332–338.
- [15] Chen, H.-Y., and Chang, Y.-W., 2009. “Global and detailed routing”. In *Electronic Design Automation*. Elsevier, pp. 687–749.
- [16] Sriram, M., Kang, S.-M., et al., 1992. “Detailed layer assignment for mcm routing”. In *International Conference on Computer Aided Design: Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, Vol. 1992, pp. 386–389.
- [17] Zhou, H., and Wong, D., 1999. “Global routing with crosstalk constraints”. *IEEE Transactions on computer-aided design of integrated circuits and systems*, **18**(11), pp. 1683–1688.
- [18] Batterywala, S., Shenoy, N., Nicholls, W., and Zhou, H., 2002. “Track assignment: A desirable intermediate step between global routing and detailed routing”. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pp. 59–66.
- [19] Wu, D., Hu, J., Mahapatra, R., and Zhao, M., 2004. “Layer assignment for crosstalk risk minimization”. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*, IEEE, pp. 159–162.
- [20] Liu, X., Zhang, Y., Yeap, G. K., Chu, C., Sun, J., and Zeng, X., 2010. “Global routing and track assignment for flip-chip designs”. In *Proceedings of the 47th Design Automation Conference*, pp. 90–93.
- [21] Vinyals, O., Fortunato, M., and Jaitly, N., 2015. “Pointer networks”. In *Advances in neural information processing systems*, pp. 2692–2700.

- [22] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S., 2016. “Neural combinatorial optimization with reinforcement learning”. *arXiv preprint arXiv:1611.09940*.
- [23] Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M., 2018. “Reinforcement learning for solving the vehicle routing problem”. In *Advances in Neural Information Processing Systems*, pp. 9839–9849.
- [24] Sutton, R. S., and Barto, A. G., 2018. *Reinforcement learning: An introduction*. MIT press.
- [25] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y., 2017. “Graph attention networks”. *arXiv preprint arXiv:1710.10903*.
- [26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., 2017. “Attention is all you need”. In *Advances in neural information processing systems*, pp. 5998–6008.
- [27] He, K., Zhang, X., Ren, S., and Sun, J., 2016. “Deep residual learning for image recognition”. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.
- [28] Ioffe, S., and Szegedy, C., 2015. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. *arXiv preprint arXiv:1502.03167*.
- [29] Whitley, D., 1994. “A genetic algorithm tutorial”. *Statistics and computing*, **4**(2), pp. 65–85.
- [30] Jing, T., Lim, M. H., and Ong, Y. S., 2003. “A parallel hybrid ga for combinatorial optimization using grid technology”. In *The 2003 Congress on Evolutionary Computation, 2003. CEC’03., Vol. 3, IEEE*, pp. 1895–1902.
- [31] Mühlenbein, H., 1992. “Parallel genetic algorithms in combinatorial optimization”. In *Computer science and operations research*. Elsevier, pp. 441–453.
- [32] Lienig, J., and Thulasiraman, K., 1993. “A genetic algorithm for channel routing in vlsi circuits”. *Evolutionary Computation*, **1**(4), pp. 293–311.
- [33] Lienig, J., 1996. “A parallel genetic algorithm for two detailed routing problems”. In *1996 IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World. ISCAS 96, Vol. 4, IEEE*, pp. 508–511.
- [34] Esbensen, H., 1994. “A macro-cell global router based on two genetic algorithms”. In *European Design Automation Conference: Proceedings of the conference on European design automation, Vol. 19*, pp. 428–433.
- [35] Karp, R. M., Vazirani, U. V., and Vazirani, V. V., 1990. “An optimal algorithm for on-line bipartite matching”. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 352–358.
- [36] Kruskal, J. B., 1956. “On the shortest spanning subtree of a graph and the traveling salesman problem”. *Proceedings of the American Mathematical society*, **7**(1), pp. 48–50.
- [37] Kastner, R., Bozorgzadeh, E., and Sarrafzadeh, M., 2002. “Pattern routing: use and theory for increasing predictability and avoiding coupling”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **21**(7), pp. 777–790.
- [38] Rivera, W., 2001. “Scalable parallel genetic algorithms”. *Artificial intelligence review*, **16**(2), pp. 153–168.
- [39] Zhou, Q., Wang, X., Qi, Z., Chen, Z., Zhou, Q., and Cai, Y., 2015. “An accurate detailed routing routability prediction model in placement”. In *2015 6th Asia Symposium on Quality Electronic Design (ASQED), IEEE*, pp. 119–122.
- [40] Chan, P. K., Schlag, M. D., and Zien, J. Y., 1993. “On routability prediction for field-programmable gate arrays”. In *Proceedings of the 30th international Design Automation Conference*, pp. 326–330.
- [41] Xie, Z., Huang, Y.-H., Fang, G.-Q., Ren, H., Fang, S.-Y., Chen, Y., and Hu, J., 2018. “Routenet: Routability prediction for mixed-size designs using convolutional neural network”. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE*, pp. 1–8.
- [42] Brown, S. D., Rose, J., and Vranesic, Z. G., 1993. “A stochastic model to predict the routability of field-programmable gate arrays”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **12**(12), pp. 1827–1838.